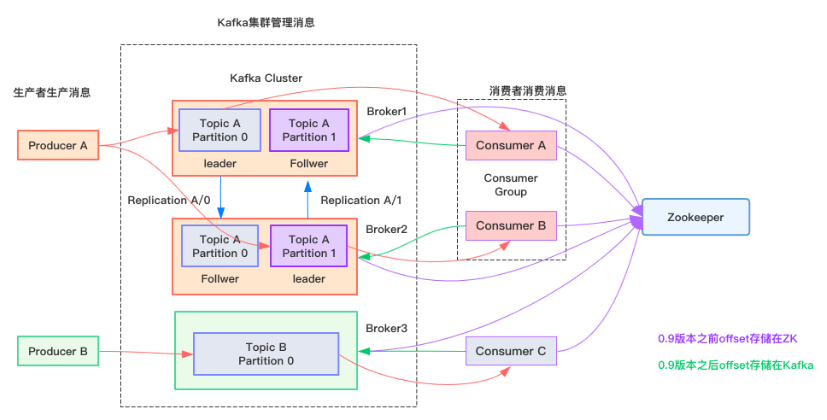
消息队列Kafka

建立日期：2021年5月20日

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本 | 变更原因 | 变更内容简述 | 编制/修订者 | 适用环境 | 发布日期 |
| V1.0 | 建立 |  | 陈文华 | 开发、测试 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Kafka是一个开源的分布式消息引擎/消息中间件，同时Kafka也是一个流处理平台。Kakfa支持以发布/订阅的方式在应用间传递消息，同时并基于消息功能添加了Kafka Connect、Kafka Streams以支持连接其他系统。

Kafka最核心的最成熟的还是他的消息引擎，所以Kafka大部分应用场景还是用来作为消息队列削峰平谷。另外，Kafka也是目前性能最好的消息中间件。



# 下载安装

1. Kafka需要jdk环境，首先配置jdk，Kafka下载地址：

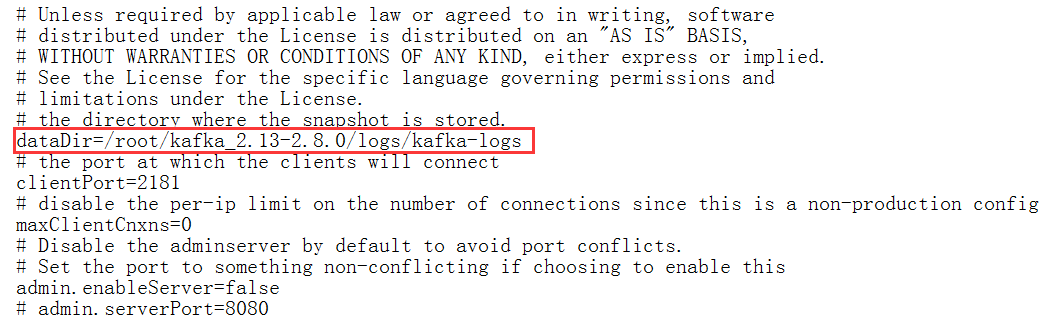
<http://kafka.apache.org/downloads>

1. 解压缩下载的二进制文件，在Kafka解压目录下创建如下两个日志目录：

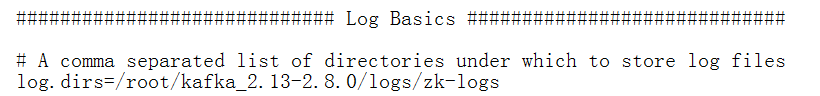
mkdir -p kafka\_2.13-2.8.0/logs/kafka-logs

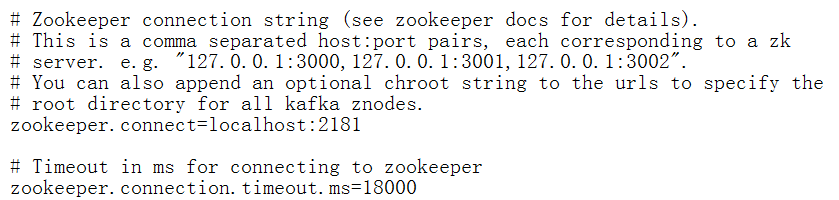
mkdir -p kafka\_2.13-2.8.0/logs/zk-logs

1. 修改config/zookeeper.properties, dataDir修改成创建的zk日志目录



修改config/server.properties, log.dirs修改成kafka日志目录





1. 启动Zookeeper

sh bin/zookeeper-server-start.sh config/zookeeper.properties

后台启动

sh bin/zookeeper-server-start.sh -daemon config/zookeeper.properties

1. 启动Kafka

sh bin/kafka-server-start.sh config/server.properties

后台启动

sh bin/kafka-server-start.sh -daemon config/server.properties

1. 创建topic

sh bin/kafka-topics.sh --zookeeper localhost:2181 --create --replication-factor 1 --partitions 1 --topic test

-create 创建

-topic 指定 topic 名

-replication-factor 定义副本数

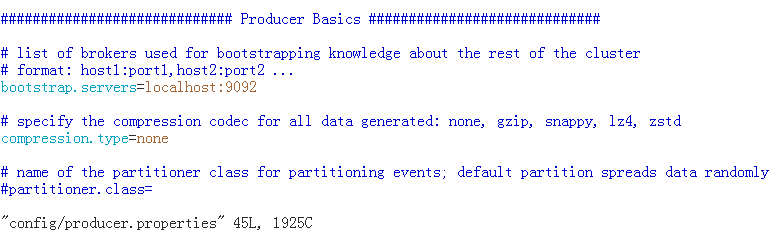
-partitions 定义分区数

上面创建了一个名称为test的topic，副本数和分区数都是1。

1. 启动生产者

sh bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

9092为生产者的默认端口号, vim config/producer.properties



生产者中增加几条数据



1. 启动消费者

sh bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning

-from-beginning：会把主题中以往所有的数据都读取出来。



1. 常用命令

查看当前服务器中的所有 topic：

sh bin/kafka-topics.sh --zookeeper localhost:2181 --list

删除 topic：

需在server.properties中设置delete.topic.enable=true否则只是标记删除。

sh bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic test

查看某个 Topic：

sh bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic test

修改分区数：

sh bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic test --partitions 2

# 搭建集群

1. 准备三台服务器

|  |  |  |
| --- | --- | --- |
| 服务器名 | IP | 说明 |
| node1 | 172.31.0.21 | Kafka节点1 |
| node2 | 172.31.0.32 | Kafka节点2 |
| node3 | 172.31.0.34 | Kafka节点3 |

1. 创建数据存储目录

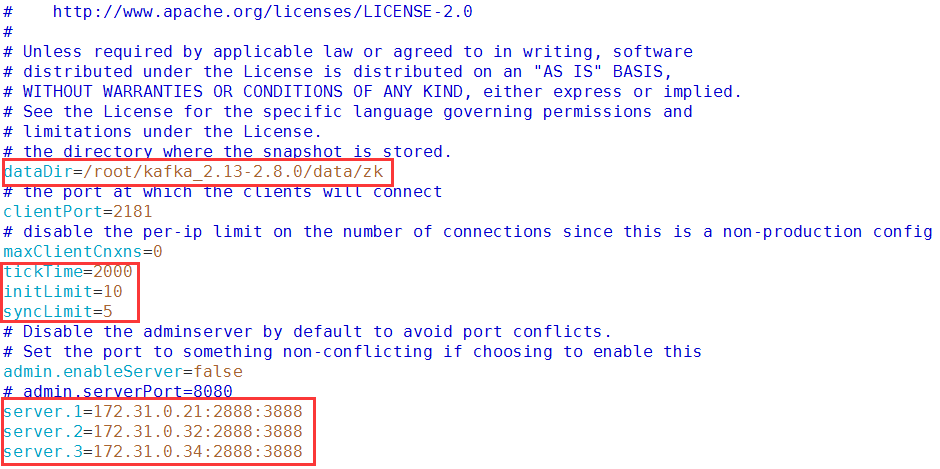
mkdir -p kafka\_2.13-2.8.0/data/zk

mkdir -p kafka\_2.13-2.8.0/data/kafka

1. 修改zk配置

cd kafka\_2.13-2.8.0

vim config/zookeeper.properties



修改添加如下内容

dataDir=/root/kafka\_2.13-2.8.0/data/zk

tickTime=2000

initLimit=10

syncLimit=5

server.1=172.31.0.21:2888:3888

server.2=172.31.0.32:2888:3888

server.3=172.31.0.34:2888:3888

三台机器上的zookeeper.properties文件配置相同，data.Dir 为zk的数据目录，server.1、server.2、server.3 为集群信息。

2888端口号是zookeeper服务之间通信的端口

3888端口是zookeeper与其他应用程序通信的端口。

tickTime：CS通信心跳数

Zookeeper 服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每个 tickTime 时间就会发送一个心跳。tickTime以毫秒为单位，该参数用来定义心跳的间隔时间，zookeeper的客户端和服务端之间也有和web开发里类似的session的概念，而zookeeper里最小的session过期时间就是tickTime的两倍。

initLimit：LF初始通信时限

集群中的follower服务器(F)与leader服务器(L)之间 初始连接 时能容忍的最多心跳数（tickTime的数量）

syncLimit：LF同步通信时限

集群中的follower服务器(F)与leader服务器(L)之间 请求和应答 之间能容忍的最多心跳数（tickTime的数量）

1. 创建myid文件

cd /root/kafka\_2.13-2.8.0/data/zk

vim myid

将三台服务器上的myid文件分别写入1，2，3。myid是zookeeper集群用来发现彼此的标识，必须创建，且不能相同。

1. 关闭防火墙

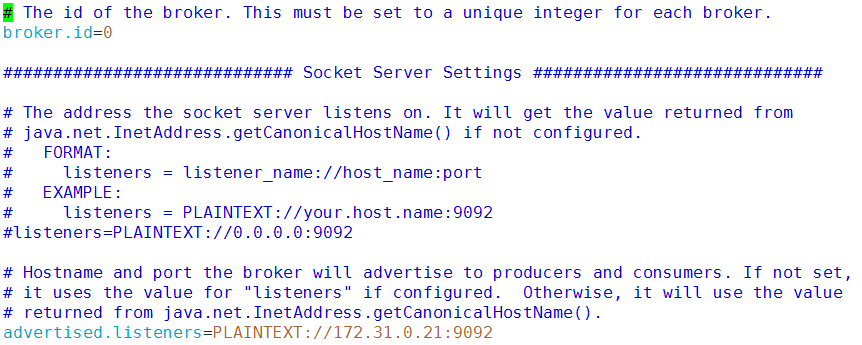
systemctl stop firewalld

sytemctl disable firewalld

1. 三台服务器都启动zk

sh bin/zookeeper-server-start.sh -daemon config/zookeeper.properties

1. 修改server.properties配置文件



broker.id=0

#broker.id的值三个节点要配置不同的值，分别配置为0，1，2

log.dirs=/root/kafka\_2.13-2.8.0/data/kafka

advertised.listeners=PLAINTEXT://172.31.0.21:9092

#三台服务器分别配置ip，172.31.0.32，172.31.0.34

#也可设置为：

listeners=PLAINTEXT://0.0.0.0:9092

advertised.listeners=PLAINTEXT://公网iP:9092

num.partitions=3

#分区数，根据需要自行修改

zookeeper.connect=172.31.0.21:2181,172.31.0.32:2181,172.31.0.34:2181

# zookeeper连接地址，多个以逗号隔开

1. 三节点都启动kafka

sh bin/kafka-server-start.sh -daemon config/server.properties

1. 创建topic

sh bin/kafka-topics.sh --zookeeper 172.31.0.21:2181,172.31.0.32:2181,172.31.0.34:2181 --create --replication-factor 1 --partitions 1 --topic test

1. 三台服务器都列出已创建的topic列表

sh bin/kafka-topics.sh --zookeeper localhost:2181 --list

1. 启动生产者增加两条测试数据

sh bin/kafka-console-producer.sh --broker-list 172.31.0.21:9092, 172.31.0.32:9092, 172.31.0.34:9092 --topic test

1. 启动消费者测试读取数据

sh bin/kafka-console-consumer.sh --bootstrap-server 172.31.0.21:9092, 172.31.0.32:9092,172.31.0.34:9092 --topic test --from-beginning

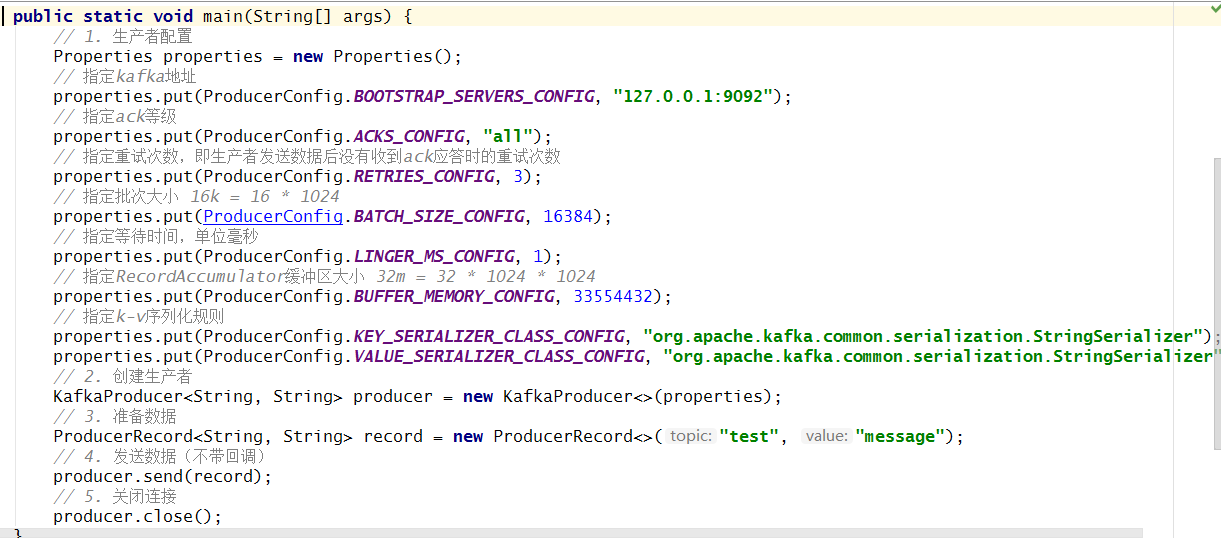
# Kafka API

创建maven项目，引入依赖

<**dependency**>  
 <**groupId**>org.apache.kafka</**groupId**>  
 <**artifactId**>kafka-clients</**artifactId**>  
 <**version**>2.8.0</**version**>  
</**dependency**>

## 3.1 生产者API

1、编写一个简单的生产者：



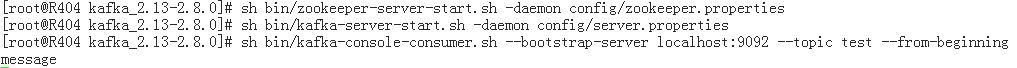
Kafka 的 Producer 发送消息采用的是异步发送的方式。在消息发送的过程中，涉及到了两个线程——main 线程和 Sender 线程，以及一个线程共享变量——RecordAccumulator。main 线程将消息发送给 RecordAccumulator，Sender 线程不断从 RecordAccumulator 中拉取消息发送到 Kafka broker。

上面代码创建了一个kv都是String类型的生产者。配置了Kafka的地址，ack等级为all，重试次数为3。并且指定了生产者发送数据的批次为16k，等待时间为1ms。也就是说，要么发送的数据量达到了16k，要么等待时间超过了1ms才会真正地把数据发往Kafka。

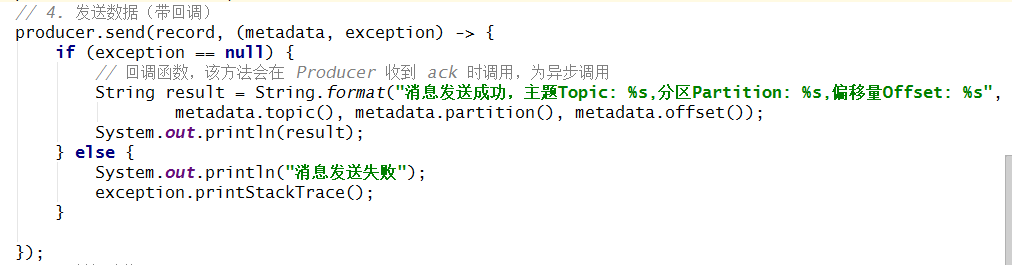
此外，我们还指定RecordAccumulator缓冲区大小，kv序列化规则采用StringSerializer。

Kafka消息使用KafkaProducer对象表示，他有6个重载构造器，这里我们指定了主题为test，消息内容为message。

启动zk，启动kafka，启动kafka消费者，运行程序



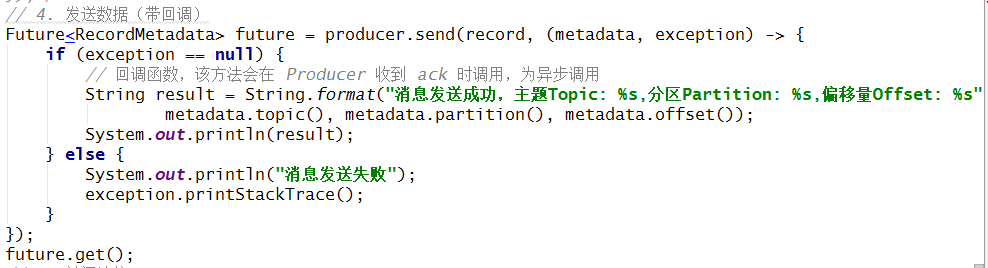
2、回调函数



KafkaProducer的send方法可以制定回调函数。回调函数会在 producer 收到 ack 时调用，为异步调用，该方法有两个参数，分别是RecordMetadata 和 Exception，如果 Exception 为 null，说明消息发送成功，如果Exception 不为 null，说明消息发送失败。

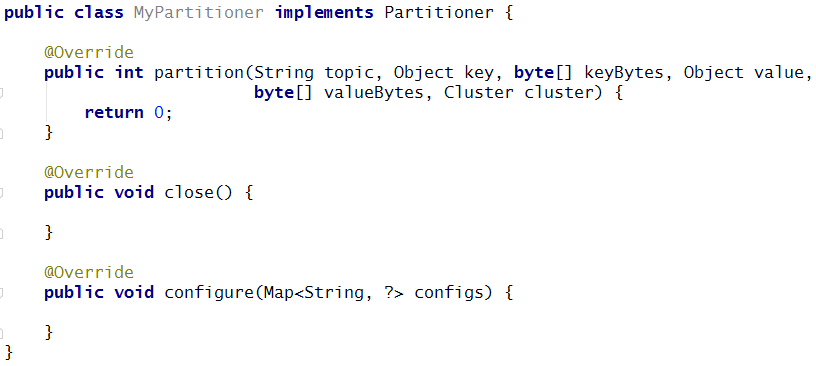
3、同步调用

Kafka 的 Producer 发送消息采用的是异步发送的方式，KafkaProducer的send方法返回Future对象，所以我们可以手动调用Future对象的get方法实现同步：



4、自定义分区器

实现Partitioner接口



要使用自定义分区器，可通过配置类指定：



## 3.2 消费者API

1、创建一个消费者，自动提交offset



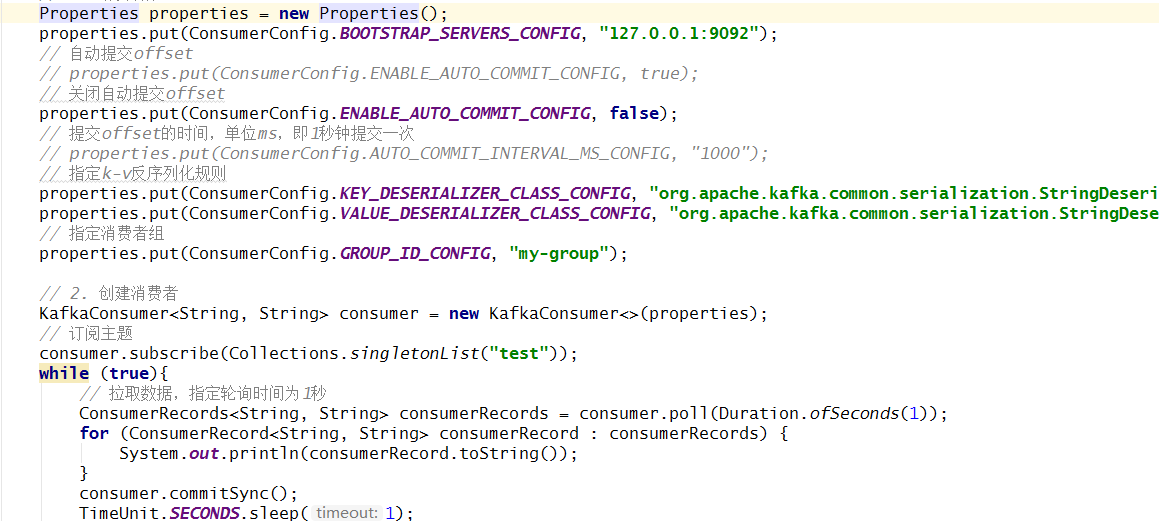
启动消费者，使用生产者发送消息到test主题

2、手动提交offset

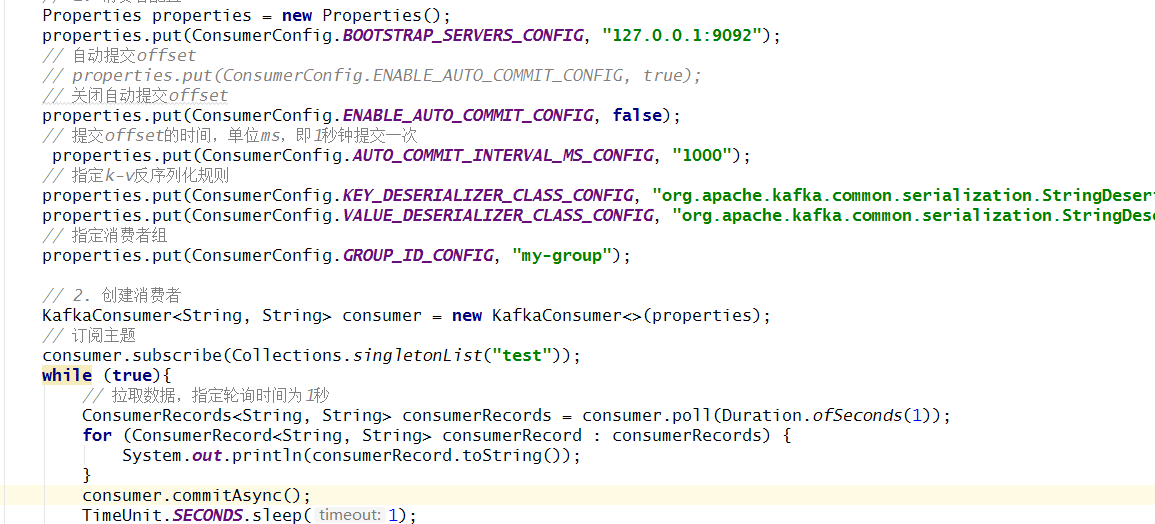
虽然自动提交 offset 十分便利，但由于其是基于时间提交的，开发人员难以把握 offset 提交的时机。因此 Kafka 还提供了手动提交 offset 的 API。

手动提交offset的方法有两种：分别是 commitSync（同步提交）和 commitAsync（异步提交）。两者的相同点是，都会将本次 poll 的一批数据最高的偏移量提交；不同点是，commitSync 阻塞当前线程，一直到提交成功，并且会自动失败重试（由不可控因素导致，也会出现提交失败）；而 commitAsync 则没有失败重试机制，故有可能提交失败。

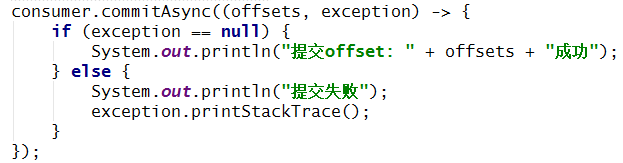
由于同步提交 offset 有失败重试机制，故更加可靠，以下为同步提交 offset 的示例：



虽然同步提交 offset 更可靠一些，但是由于其会阻塞当前线程，直到提交成功。因此吞吐量会收到很大的影响。更多的情况下，会选用异步提交 offset 的方式：



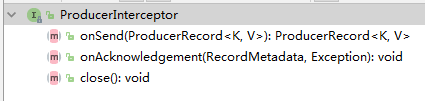
commitAsync()可以指定回调：



## 3.3 拦截器

1、Producer 拦截器(interceptor)使得用户在消息发送前以及 producer 回调逻辑前有机会对消息做一些定制化需求，比如修改消息等。实现接口是

org.apache.kafka.clients.producer.ProducerInterceptor，其定义的方法包括：



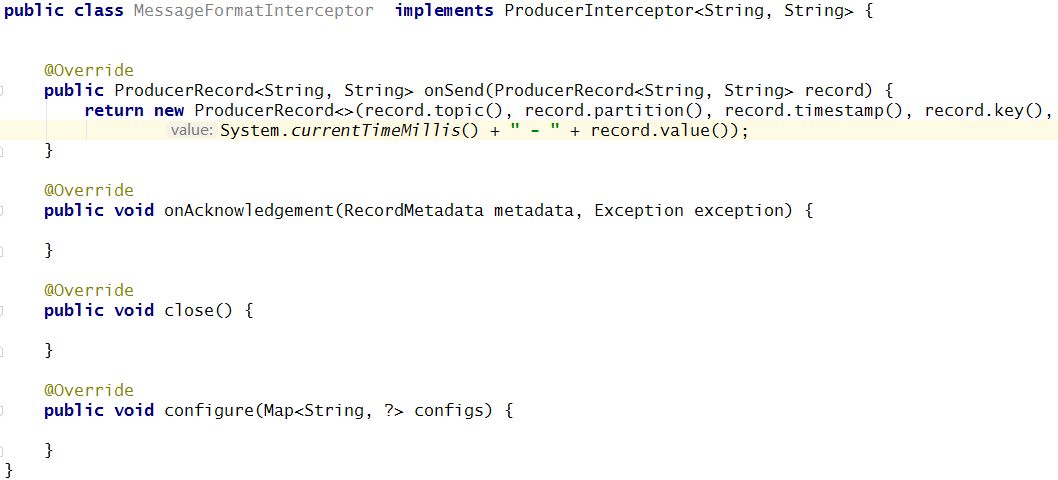
configure(configs)：获取配置信息和初始化数据时调用；

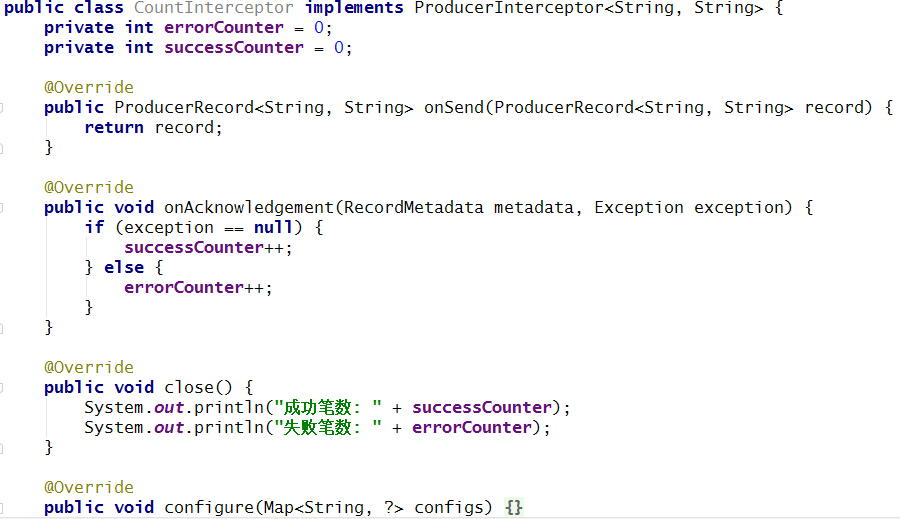
onSend(ProducerRecord)：在消息被序列化以及计算分区前调用该方法。用户可以在该方法中对消息做任何操作，但最好不要修改消息所属的 topic 和分区，否则会影响目标分区的计算；

onAcknowledgement(RecordMetadata, Exception)：在返回ack时，或者发送失败时调用该方法。

close：关闭 interceptor，主要用于执行一些资源清理工作。

下面举个自定义拦截器的例子：定义一个拦截器链，包含两个拦截器MessageFormatInterceptor和CountInterceptor，分别用于消息加工和统计消息发送成功和失败的笔数。





在生产者配置中指定这个过滤器链：



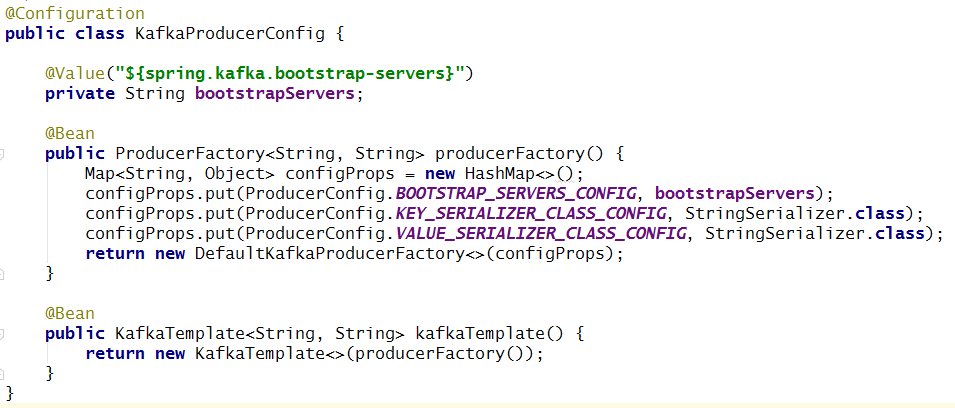
# 四、Springboot整合Kafka

创建springboot项目，添加依赖

<**dependency**>  
 <**groupId**>org.springframework.kafka</**groupId**>  
 <**artifactId**>spring-kafka</**artifactId**>  
</**dependency**>

## 配置生产者

创建一个配置类，用于配置生产者：



bootstrapServers为Kafka生产者的地址，在application.yml配置



## 2、发布消息

创建一个controller



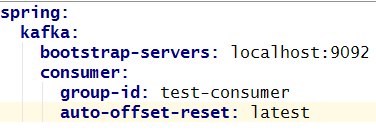
## 3、配置消费者



配置类上需要@EnableKafka注释才能在Spring托管Bean上检测到

@KafkaListener注解，consumerGroupId和autoOffsetReset需要在

application.yml里配置：



## 4、消息消费

创建KafkaMessageListener



通过@KafkaListener注解来监听名称为test的Topic，消费者分组的组名为test-consumer

## 5、启动项目

访问 <http://localhost:8080/send/hello>

控制台输出

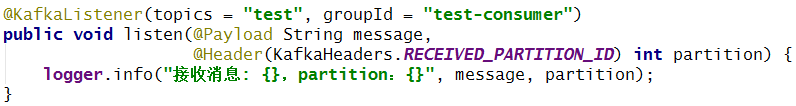


## 6、@KafkaListener注解

* @KafkaListener除了可以指定Topic名称和分组id外，我们还可以同时监听来自多个Topic的消息:

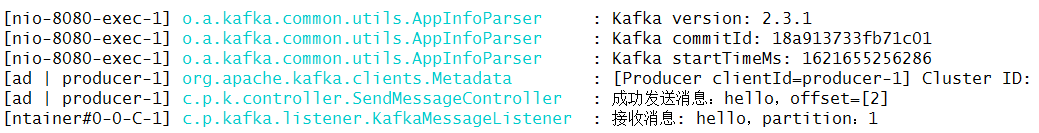
@KafkaListener(topics = "topic1, topic2")

* 通过@Header注解来获取当前消息来自哪个分区（partitions）:



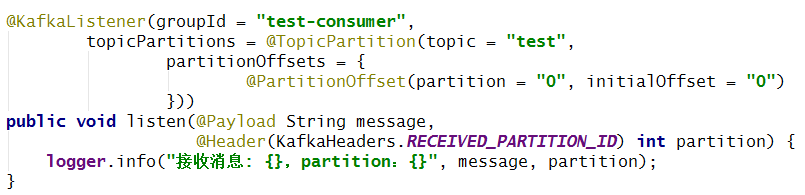
重启项目再次访问<http://localhost:8080/send/hello>

控制台输出

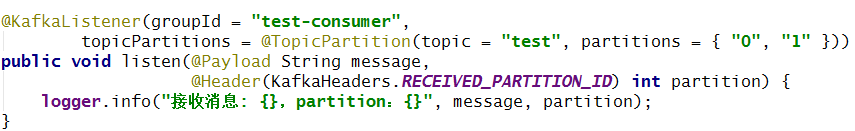


因为没有进行分区，所以test Topic只有一个区，下标为0。

* 通过@KafkaListener来指定只接收来自特定分区的消息：

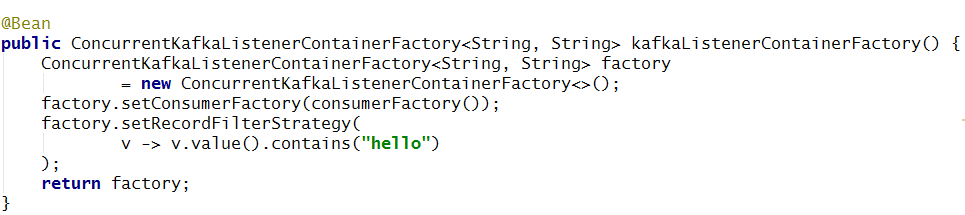


如果不需要指定initialOffset，上面代码可以简化为：



## 7、消息过滤器

在消费者配置类KafkaConsumerConfig的kafkaListenerContainerFactory方法里配置过滤规则：



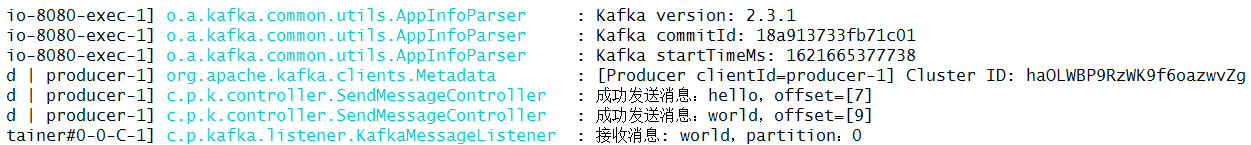
即如果消息内容包含hello的时候，则不接受消息。

重启项目，分别发送下面这两条请求

<http://localhost:8080/send/hello>

<http://localhost:8080/send/world>

控制台输出：



## 8、发送复杂消息

* 定义消息实体，创建Message类



* 修改消息生产者配置，将value序列化策略指定为Kafka的JsonSerializer，并且kafkaTemplate返回类型为KafkaTemplate<String, Message>



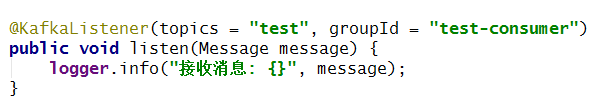
* 修改controller，发送复杂的消息：



* 修改消费者配置：



* 修改消息监听器：



* 重启项目，访问：

<http://localhost:8080/send/message>

控制台输出

